AFRL-AFOSR-UK-TR-2010-0016

Advanced Non-Equilibrium Modelling for Hypersonic Applications

Olivier P Chazot

Von Karman Institute for Fluid Dynamics (VKI)
72 Chaussee de Waterloo
Rhode-Saint-Genese, Belgium B-1640

EOARD GRANT 08-3070

January 2011

Final Report for 01 August 2008 to 01 August 2010

Air Force Research Laboratory
Air Force Office of Scientific Research
European Office of Aerospace Research and Development
Unit 4515 Box 14, APO AE 09421

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
26 January 2011	Final Report	01 August 2008 – 01 August 2010

4. TITLE AND SUBTITLE

Advanced Non-Equilibrium Modelling for Hypersonic Applications

5a. CONTRACT NUMBER

FA8655-08-1-3070

5b. GRANT NUMBER

Grant 08-3070

5c. PROGRAM ELEMENT NUMBER

61102F

6. AUTHOR(S)

Professor Olivier Chazot

5d. PROJECT NUMBER

5d. TASK NUMBER

5e. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Von Karman Institute for Fluid Dynamics (VKI)
72 Chaussee de Waterloo
Rhode-Saint-Genese B-1640
Belgium

8. PERFORMING ORGANIZATION REPORT NUMBER

Grant 08-3070

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
Unit 4515 BOX 14
APO AE 09421

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/AFOSR/RSW (EOARD)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

AFRL-AFOSR-UK-TR-2010-0016

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In this final report we provide a detailed account of how to call the subroutines of the Mutation library in the context of a practical computational fluid dynamics (CFD) application. The subroutines which were documented in the previous report are shown how to be called in a detailed, step-by-step fashion, from computation of the thermodynamic properties to the transport properties. Specific emphasis is given to integrating the Mutation library with external CFD solvers provided by the user. This is accomplished by providing detailed source code which shows exactly how the library subroutines are to be called. The source code is fully-functioning and may serve as a template for users to build their solvers around that will enable researchers working with the Mutation library to integrate their applications with ease. The subroutines and example code herein, provided in three different computing languages, are a direct implementation of the theory discussed in the second report. Every attempt has been made to use consistent nomenclature and notation, so that the code provided here can be easily compared with the theoretical description.

15. SUBJECT TERMS

EOARD, Thermal Protection & Control, Plasma Aerodynamic, Hypersonic Flow

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18, NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Gregg Abate
UNCLAS	UNCLAS	UNCLAS	SAR	33	19b. TELEPHONE NUMBER *(Include area code)* +44 (0)1895 616021

Advanced non-equilibrium modelling for hypersonic applications

Investigators: Prof. Olivier Chazot (P.I.), Prof. Thierry E. Magin, Dr. M. Panesi, and Dr. M. Kapper

Olivier.Chazot@vki.ac.be

Aeronautics and Aerospace Department

von Karman Institute for Fluid Dynamics

Contents

This Page is Blank

1 Introduction

In this final report we provide a detailed account of how to call the subroutines of the Mutation library in the context of a practical computational fluid dynamics (CFD) application. The subroutines which were documented in the previous report are shown how to be called in a detailed, step-by-step fashion, from computation of the thermodynamic properties to the tranport properties. Specific emphasis is given to integrating the Mutation library with external CFD solvers provided by the user. This is accomplished by providing detailed source code which shows exactly how the library subroutines are to be called. The source code is fully-functioning and may serve as a template for users to build their solvers around that will enable researchers working with the Mutation library to integrate their applications with ease.

The subroutines and example code herein, provided in three different computing languages, are a direct implementation of the theory discussed in the second report. Every attempt has been made to use consistent nomenclature and notation, so that the code provided here can be easily compared with the theoretical description.

2 Conservation equations for reactive and plasma flows

A previously discussed, the Mutation library is intended to provide thermodynamic, chemical production rates and transport properties to multi-component plasma flows in high-enthalpy flows such as re-entry and hypersonic, air-breathing vehicles. In a typical CFD application for flows in the continuum regime, the Navier-Stokes equations can be solved via any one of numerous available numerical techniques. As an example, when solving for the flow structure through shock-heated plasma, approximate Riemann solvers can be employed to solve for the convective fluxes when shock-capturing is desired. If shock-capturing techniques are not required, the shock jump conditions can be computed from the Rankine-Hugoniot relations, which provide the initial conditions for solving the system of ordinary differential equations that result from reducing the Navier-Stokes equations to their time-independent form. Whichever solution technique is used to solve the convective component of the governing equations, it is the subroutines of the Mutation library which compute the thermodynamic and transport properties of the flow. The task of interfacing a CFD application and the Mutation library simply becomes a matter of passing the flow variables from the CFD solver to the Mutation subroutines.

To illustrate how the interfacing is done, the Navier-Stokes equations in the following sections are reduced to three simplified cases for which the Mutation library provides specialized subroutines. The subroutines are detailed in the text and correspond directly to the working code examples provided in Appendix A. Working code is provided in Fotran 77, C, and Java.

2.1 Species continuity

For multi-component flows, the continuity equation for the i^{th} specie may be written as

$$\frac{\partial \rho_i}{\partial t} + \nabla \cdot (\rho_i \mathbf{v}) + \nabla \cdot (\rho_i \mathbf{V}_i) = M_i \dot{\omega}_i, \quad i \in \mathcal{S} \tag{1}$$

where ρ_i is the mass density of specie i, $\mathbf{v}$ is the bulk velocity, and $\mathbf{V}_i$, $\mathbf{M}_i$, and $\dot{\omega}_i$ are respectively the mass diffusion velocity, molecular weight, and mass production rate of species i.

2.2 Total continuity

Summing equation (1) over all species, the total mass continuity equation is obtained

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{2}$$

where $\rho = \sum_{i \in \mathcal{S}} \rho_i$ is the total mass density, $\sum_{i \in \mathcal{S}} \rho_i \mathbf{V}_i = 0$ and $\sum_{i \in \mathcal{S}} M_i om\dot{e}ga_i = 0$.

2.3 Momentum equation

The momentum equation is given by

$$\frac{\partial \rho \mathbf{v}}{\partial t} + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v}) + \nabla \cdot \mathsf{P} - nq\mathbf{E}' - \mathbf{j} \times \mathbf{B} = 0 \tag{3}$$

where P is the stress tensor, $q = \sum_{i \in \mathcal{S}} x_i q_i$ is the mixture charge, x_i is the mole fraction, and $\mathbf{j} = \sum_{i \in \mathcal{S}} n_i q_i \mathbf{V}_i$ is the mixture conduction current. The electric field in the hydrodynamic field is given by $\mathbf{E}' = \mathbf{E} + \mathbf{v} \times \mathbf{B}$.

2.4 Species energy conservation

The species energy equations are given by

$$\frac{\partial \rho_i e_i}{\partial t} + \nabla \cdot (\rho_i e_i \mathbf{v}) + \nabla \cdot \mathbf{q}_i - \mathbf{j}_i \cdot \mathbf{E}' + \rho_i \mathbf{V}_i \cdot \frac{d}{dt}\mathbf{v} + \mathsf{P}_i : \nabla \mathbf{v} = \Delta E_i, \quad i \in \mathcal{S} \tag{4}$$

for species energy $\rho_i e_i = \frac{3}{2} n_i k_B T_i$ for translational modes, with the species heat flux $\mathbf{q}_i$, and energy exchange terms ΔE_i. For molecular plasma, ρe_{rot} and ρe_{vib} may also be convected for cases under thermal non-equilibrium where the translational, rotational, and vibrational temperatures vary greatly from one another.

2.5 Total energy conservation

Summing equation (4) over all species, the total energy conservation equation is obtained

$$\frac{\partial E}{\partial t} + \nabla \cdot (E\mathbf{v}) + \nabla \cdot \mathbf{q} - \mathbf{j} \cdot \mathbf{E}' + \mathsf{P} : \nabla \mathbf{v} = 0 \tag{5}$$

where $E = \rho e = \sum_{i \in \mathcal{S}} \rho_i e_i$ is the total energy density and $\mathbf{q} = \mathbf{q}_h + \mathbf{q}_e$ is the total heat flux (sum of heavy-particle and electron components).

3 Cases: thermodynamic and transport properties

The Mutation library is applicable to solving distinct cases of the governing equations above, for which three possible applications are reviewed here. They employ various levels of simplifying assumptions which allow a range of computational performance balanced with accuracy. These cases are presented in the following sections, whose governing equations will be given in the form

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}^c}{\partial x} + \frac{\partial \mathbf{F}^d}{\partial x} = \mathbf{S} \tag{6}$$

where $\mathbf{Q}$ is the vector of conserved variables, $\mathbf{F}^c$ is the vector of convective fluxes, $\mathbf{F}^d$ is the vector of diffusive fluxes, and $\mathbf{S}$ is the vector of source terms. It is important to note that throughout this report, only the conservative form of the governing equations are considered which are the most general and must be used when solving for time-accurate flows. However, if desired, the Mutation library can be applied to primitive variable formulations and is straight-forward.

3.1 Local thermodynamic equilibrium (LTE) with constant elemental fractions

When the assumption of chemical equilibrium with constant elemental fractions can be made, the governing equations can be greatly simplified. Only the total species continuity equation (2) need be solved as the individual species mass densities can be obtained based on equilibrium considerations. Furthermore, under local thermal equilibrium (LTE), the various internal modes are in equilibrium and the total energy is obtained from (5). The governing equations reduce to

$$\mathbf{Q} = [\rho, \rho\mathbf{v}, E]^T \tag{7}$$

$$\mathbf{F^c} = [\rho\mathbf{v}, \rho\mathbf{v} \otimes \mathbf{v} + p\mathsf{I}, \mathbf{v}H]^T \tag{8}$$

$$\mathbf{F^d} = [0, \tau + \mathbf{q}, \mathbf{v}\tau]^T \tag{9}$$

$$\mathbf{S} = [0, F^L, \sigma_e < E^2 >]^T \tag{10}$$

where I is the identity matrix, H is the total enthalpy density, τ is the deviatoric stress tensor, and $\sigma_e < E^2 >$ are the energy-averaged thermal relaxation terms.

In this case, the following subroutines provided in the Mutation library can be used to compute the thermodynamic properties of the flow, including the bulk temperature and pressure once the conservative variables are provided by the CFD code. the methods called in order to compute the thermodynamic and transport properties of the fluid.

- Compute elemental mole fractions
  `CALL NUCLEAR (XN)`
- Compute bulk temperature and species mass fractions
  `CALL TCEQNEWTON (RHOE, RHO, XN, YINI, TINI, T, Y)`
- Compute mole fractions from mass fractions
  `CALL MASS2MOLE (Y, X)`

- Compute pressure using ideal gas law
  `CALL MASS2PRESSURE (RHO, T, T, Y, P)`

- Compute the total number density
  `CALL NUMBERD (P, T, T, X, ND)`

With the exception of `CALL NUCLEAR (XN)`, theses subroutines are called at every iteration in a CFD computation. Good guesses for the initial temperature, mole and mass fractions can be obtained from the previous CFD interation. A full description of the subroutine parameters are provided in the Appendix B.

3.2 Chemical non-equilibrium and thermal equilibrium

A second case detailed here, is applicable to flows in LTE but in chemical non-equilibrium. In such a case, the species continuity equations are solved for each species, thus increasing the size of the system by the number of species and requiring computation of the mass diffusion fluxes. Furthermore, finite chemistry requires the rates of chemical production for both mass and energy.

$$\mathbf{Q} = [\rho_i, \rho\mathbf{v}, E]^T \tag{11}$$

$$\mathbf{F^c} = [\rho\mathbf{v}, \rho\mathbf{v} \otimes \mathbf{v} + p\mathbf{I}, \mathbf{v}H]^T \tag{12}$$

$$\mathbf{F^d} = [\rho_i\mathbf{V}_i, \tau + \mathbf{q}, \mathbf{v}\tau]^T \tag{13}$$

$$\mathbf{S} = [M_i\dot{\omega}_i, F^L, \dot{\omega}_i H_i^\circ]^T \tag{14}$$

where M_i are the molar masses of species i and H_i° are the formation energies.
In the case the subroutines to be called are

- Compute bulk temperature
  `CALL TCNEQNEWTON (RHOE, RHOI, TINI, T)`

- Compute mole fractions from mass fractions
  `CALL MASS2MOLE (Y, X)`

- Compute pressure using ideal gas law
  `CALL MASS2PRESSURE (RHO, T, T, Y, P)`

- Compute the total number density
  `CALL NUMBERD (P, T, T, X, ND)`

- Compute the rates for chemistry
  `CALL ARRHENIUS (Y, TOLY, P, TVEC, RHO, OMEGA)`

The subroutines are similar to the first case with the exception of a call to `ARRHENIUS` which provides the chemical rates of production. The Newton solver obtains the bulk temperature implicitly from the mass densities and total energy density.

3.3 Chemical and thermal non-equilibrium ($T_h \neq T_e \neq T_r \neq T_v$)

Finally, we consider the case of both chemical and thermal non-equilibrium. The previous case is augmented to include multiple energy equations for each internal energy component which cannot be considered in equilibrium with any other component.

$$\mathbf{Q} = [\rho_i, \rho\mathbf{v}, \rho_i e_i]^T \tag{15}$$

$$\mathbf{F^c} = [\rho\mathbf{v}, \rho\mathbf{v} \otimes \mathbf{v} + p\mathbf{I}, \mathbf{v}H_i]^T \tag{16}$$

$$\mathbf{F^d} = [\rho_i \mathbf{V}_i, \tau + \mathbf{q}, \mathbf{v}\tau_i]^T \tag{17}$$

$$\mathbf{S} = [M_i\dot{\omega}_i, F^L, \sigma_{e_i} < E_i^2 >]^T \tag{18}$$

In the case the subroutines to be called are

- Compute component temperatures
  `CALL TCNEQNTNEWTON (RHOE, RHOEE, RHOER, RHOEV, RHOI, TINI, TH, TE, TR, TV)`
- Compute mole fractions from mass fractions
  `CALL MASS2MOLE (Y, X)`
- Compute pressure using ideal gas law
  `CALL MASS2PRESSURE (RHO, T, T, Y, P)`
- Compute the total number density
  `CALL NUMBERD (P, T, T, X, ND)`
- Compute the rates for chemistry
  `CALL ARRHENIUS (Y, TOLY, P, TVEC, RHO, OMEGA)`

The routine `TCNEQNTNEWTON` takes into account the various energies.

3.4 Caloric equation of state (EOS)

As currently implemented in the Mutation library, the specific heats are computed via finite differences based on an appropriate ΔT. The procedure implemented in the example code is summed up as follows.

- Compute perturbed temperatures for finite differences
  `EPS1 = 1.D0 + EPS, THP = TH*EPSP1, TEP = TE*EPSP1, ...`
- Compute the species specific internal energies
  `CALL ENERGYMASSF(TH , TE , TR , TV , P, EM1 , EM2 , EM3 , EM4 , EM5 , EM6)`
  `CALL ENERGYMASSF(THP, TEP, TRP, TVP, P, EM1P, EM2P, EM3P, EM4P, EM5P, EM6P)`
- Compute the specific heat for internal modes
  `CPE` $= \sum_I$ `((EM3P(I) - EM3(I))/(EPS*TH))`
  `CPR` $= \sum_I$ `((EM4P(I) - EM4(I))/(EPS*TH))`
  `CPV` $= \sum_I$ `((EM5P(I) - EM5(I))/(EPS*TH))`
  `CPINT(1:NS) = MMI(1:NS)*(CPE + CPR + CPV)`

Here, `1:NS` represents the vector of all species from 1 to the number of species. Current efforts are aimed at increasing the efficiency of this computation by pre-tabulating the specific heats to generate look-up tables that can then be accessed by the subroutines.

3.5 Transport properties

With thermodynamic properties and chemical production rates computed, the following subroutines can be used to compute the transport properties including viscosity and conductivity for all 3 cases mentioned above. The specific applicability of each subroutine is as indicated.
Firstly,

- Compute collision integrals
  `CALL COLLISION (TH, TE, ND, X)`

- Compute tolerances according to mass constraint
  `CALL COMPOTOL (X, TOLX, XTOL)`

- Compute the mixture viscosity using conjugate gradient method
  `CALL ETACG (XTOL, ETA)`

- Compute the internal thermal conductivity using Eucken's formula
  `CALL LAMBDAINT (CPINT, XTOL, LAMBDAINTH)`

- Compute the translational thermal conductivity and thermal diffusion ratios of heavy-particle gas
  `CALL LAMBDACHICG (XTOL, LAMBDATH, CHIH)`

If the mixture contains electrons then the properties of the electron gas are computed via

- Compute the thermal diffusion ratios of the electron gas
  `CALL TDIFE (XTOL, TE, CHIE, 3)`

- Compute the translational thermal conductivity of the electron gas
  `CALL LAMBDAE (XTOL, TE, LAMBDATE, 3)`

Conversely, if the mixture does not contain electron, then these values are set to null,

- Electron translational thermal conductivity is null
  `LAMBDATE = 0`

- Electron thermal diffusion ratios are null
  `CHIE(1:NS) = 0`

3.5.1 Diffusion fluxes

The diffusion fluxes may be computed in one of two ways, using either Stefan-Maxwell's equations or Fick's law with Ramshaw's approximations for small electron mass. For mixtures with charged particles, the following subroutine solves the Ramshaw system to obtain the diffusion fluxes.

- Compute the correction functions of the Stefan-Maxwell equations of the heavy particle gas
  `CALL CORRECTION (XTOL, 1, FIJ)`
- Compute the mass diffusion fluxes via solution of the Stefan-Maxwell equations (charged particles)
  `CALL SMRAMCG (XTOL, TH, TE, ND, DF, FIJ, JDIF, EAMB)`

for neutral mixtures, the appropriate calls are

- Compute the correction functions of the Stefan-Maxwell equations of the heavy particle gas
  `CALL CORRECTION (XTOL, 1, FIJ)`
- Compute the mass diffusion fluxes solution of the Stefan-Maxwell equations (neutral mixture)
  `CALL SMNEUTCG (XTOL, ND, DF, FIJ, JDIF)`
- Ambipolar electric field is null
  `EAMB = 0`

In either case, the driving forces may be computed by

```
DF(1:NS) = (XP(1:NS) - X(1:NS))/(TH*EPS) + (CHIE(1:NS) + CHIH(1:NS))/TH
```

Since the Ramshaw approximation yields an implicit system, it may be desirable to compute the diffusion fluxes from the generalized Fick's law when Jacobian matrices must be computed explicitly,

- Compute the binary diffusion coefficient using generalized Fick's law
  `CALL DIJFICK (XTOL, ND, DIJ)`
- Compute the diffusion fluxes
  `JDIF(1:NS)` $= \sum_J$ `(JDIF(1:NS) - RHOI(1:NS)*DIJ(1:NS,J)*DF(J))`

3.5.2 Thermal conductivity

Correct calculation of the total thermal conductivity is dependent upon the equilibrium state of the mixture. In LTE, the following routine can be used

- Compute the total conductivity for LTE mixture
  `CALL KCONDUCTIVITY (P, TH, X, XTOL, XN, EPS, LAMBDATOT)`

Otherwise, the total thermal conductivity must be computed from its individual contributions

$$\texttt{LAMBDAR} = -\sum_{I} (\texttt{(EM1(I) + 2/3*EM2(I))*JDIF(I)})$$

```
LAMBDATOT = LAMBDAINTH + LAMBDATH + LAMBDATE + LAMBDAR
```

where `LAMBDAR` is the reactive thermal conductivity based on the Stefan-Maxwell equations and is valid in chemical equilibrium as well.
For the case of chemical non-equilibrium with thermal equilibrium, the following subroutine can be called

- Compute the total thermal conductivity in chemical non-equilibrium and thermal equilibrium
  `KCONDUCTIVITYNEQ (P, TH, X, XTOL, XN, EPS, LAMBDATOT)`

3.6 Deallocating memory

A convenience method has been provided to automatically deallocate all memory utilized by the Mutation library. The following routine can be called as such

- Deallocate all memory
  `CALL FINALIZE()`

4 Documentation and distribution

The same subroutines are available on all programming platforms and include required input with indicated units along with the returned values. Although, the documentation provided here has been generated from the Java interface, subroutine calls for the most part remain identical in Fortran and C variants. This version supercedes previous versions.

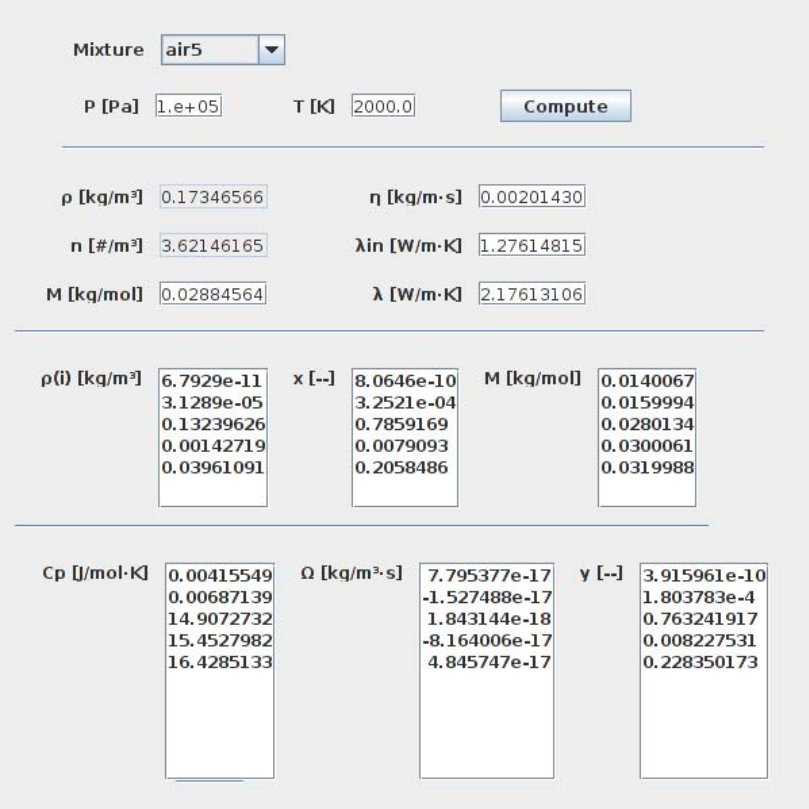

Figure 1: Java applet as it will appear on Mutation's home page at vki.ac.be.

As the Mutation library undergoes modification and expansion, necessary updates and revisions will follow. To ensure availability and consistency of the latest releases, a revision control will be maintained online with the web address to be disclosed. From this site users will be able to download the most recent Mutation library along with documentation. Also available will be an interactive applet that will allow users to run Mutation remotely. A representative snapshot of the applet GUI is provided in Fig. 1.

A Source code

A.1 Fortran 77 implementation

```
C
      PROGRAM USEMUTATION
C
      CHARACTER*31 PATH
      CHARACTER*4  MIXTURE
      CHARACTER*4  REACTION
      CHARACTER*5  TRANSFER

      INTEGER NSMAX,IMOD
      INTEGER IS,IC,IV,IE,IVIB,IELEQ
      INTEGER NS,NC,NV,NE,NVIB,NELEQ

      PARAMETER (NSMAX = 5)

      DOUBLE PRECISION EPS, EPSP1
      DOUBLE PRECISION TOLX, TOLY
      DOUBLE PRECISION XTOL(1:NSMAX)
      DOUBLE PRECISION RHO, RHOE, RHOEE, RHOEV(1:NSMAX)
      DOUBLE PRECISION MM, MMI(1:NSMAX)
      DOUBLE PRECISION XN(1:NSMAX)
      DOUBLE PRECISION XP(1:NSMAX)
      DOUBLE PRECISION X(1:NSMAX), Y(1:NSMAX)
      DOUBLE PRECISION Y3(1:NSMAX), Y4(1:NSMAX), Y5(1:NSMAX)
      DOUBLE PRECISION XINI(1:NSMAX), YINI(1:NSMAX)
      DOUBLE PRECISION RHOI(1:NSMAX)
      DOUBLE PRECISION P, E, ND
      DOUBLE PRECISION T, T2, T3, TINI
      DOUBLE PRECISION TH , TE , TR , TV (1:NSMAX)
      DOUBLE PRECISION THP, TEP, TRP, TVP(1:NSMAX)
      DOUBLE PRECISION TVEC(1:NSMAX+2)
      DOUBLE PRECISION EM, HM
      DOUBLE PRECISION EM1 (1:NSMAX), EM2 (1:NSMAX), EM3 (1:NSMAX)
      DOUBLE PRECISION EM4 (1:NSMAX), EM5 (1:NSMAX), EM6 (1:NSMAX)
      DOUBLE PRECISION EM1P(1:NSMAX), EM2P(1:NSMAX), EM3P(1:NSMAX)
      DOUBLE PRECISION EM4P(1:NSMAX), EM5P(1:NSMAX), EM6P(1:NSMAX)
      DOUBLE PRECISION HM1 (1:NSMAX), HM2 (1:NSMAX), HM3 (1:NSMAX)
      DOUBLE PRECISION HM4 (1:NSMAX), HM5 (1:NSMAX), HM6 (1:NSMAX)
      DOUBLE PRECISION OMEGA(1:NSMAX)
      DOUBLE PRECISION ETA
      DOUBLE PRECISION CPE, CPR, CPV
      DOUBLE PRECISION CPINT(1:NSMAX)
      DOUBLE PRECISION LAMBDAINTH, LAMBDATH, LAMBDATE
      DOUBLE PRECISION CHIE(1:NSMAX), CHIH(1:NSMAX)
      DOUBLE PRECISION FIJ(1:NSMAX*(NSMAX-1)/2), JDIF(1:NSMAX)
```

```
      DOUBLE PRECISION DF(1:NSMAX)

C-----------------------------------------------------------------------
      PATH = '../mutation/resources'
      MIXTURE = 'air5'
      REACTION = 'air5'
      TRANSFER = 'empty'

      IMOD = 0

      EPS    = 1.D-2
      EPSP1 = 1.D0 + EPS

      TOLX = 1.D-16
      TOLY = 1.D-20

C-----------------------------------------------------------------------
      CALL LENGTHF(PATH,MIXTURE,REACTION,TRANSFER)

      NS     = GETNS()
      NC     = GETNC()
      NE     = GETNE()
      NV     = GETNV()
      NVIB   = GETNVIB()
      NELEQ = GETNELEQ()

C-----------------------------------------------------------------------
      TE = TH;
      TR = TH;
      IF( NV == 0 ) THEN
        TV(1) = 1.D0
      ELSE
        DO IV = 1, NV
          TV(IV) = TH
        ENDDO
      ENDIF

      TVEC(1) = TH
      DO IVIB = 1, NVIB
        TVEC(1+IVIB) = TV(1)
      ENDDO
      TVEC(NVIB+2) = TE

C-----------------------------------------------------------------------
      THP = TH*EPSP1
      TEP = TE*EPSP1
      TRP = TR*EPSP1
      IF( NV == 0 ) THEN
        TVP(1) = TV(1)*EPSP1
```

```
      ELSE
        DO IV = 1, NV
          TVP(IV) = TV(IV)*EPSP1
        ENDDO
      ENDIF

C-----------------------------------------------------------------------
      DO IS = 1, NS
        X(IS) = 1.D0/NS; Y(IS) = 1.D0/NS
        XINI(IS) = X(IS); YINI(IS) = Y(IS)
        Y4(IS) = 1.D0/NS
      ENDDO


C-----------------------------------------------------------------------
C       Subroutines to be called only once
C-----------------------------------------------------------------------
      CALL INITIALIZEF(IMOD)
      CALL SETMASSF(MMI)
      CALL NUCLEARF(XN)


C-----------------------------------------------------------------------
C       CASE 1: Local thermodynamic equilibrium (LTE)
C               with constant elemental fractions
C
C       -Obtain RHO, RHOE from CFD code
C       -Use temperatue and specie mass fractions from previous
C        iteration as initial guess
C-----------------------------------------------------------------------
      CALL TCEQNEWTONF(RHOE, RHO, XN, YINI, TINI, T, Y)
      CALL MASS2PRESSUREF(RHO, T, T, Y, P)
      CALL MASS2MOLEF(Y, X)
      CALL NUMBERDF(P, T, T, X, ND)

      TH = T; TE = TH; TR = TH
      TVEC(1) = TH
      TVEC(NVIB+2) = TE


C-----------------------------------------------------------------------
C       CASE 2: Chemical non-equilibrium and
C               local thermal equilibrium (LTE)
C
C       -Obtain RHO, RHOI, Y, RHOE from CFD code
C       -Use temperatue and specie mass fractions from previous
C        iteration as initial guess
C-----------------------------------------------------------------------
      CALL TCNEQNEWTONF(RHOE, RHOI, TINI, TH)
```

```
      CALL MASS2PRESSUREF(RHO, TH, TH, Y, P)
      CALL MASS2MOLEF(Y, X)
      CALL NUMBERDF(P, TH, TH, X, ND)

      TE = TH; TR = TH
      TVEC(1) = TH
      TVEC(NVIB+2) = TE

      CALL ARRHENIUSF(Y, TOLY, P, TVEC, RHO, OMEGA)


C-----------------------------------------------------------------------
C     CASE 3: Chemical and thermal non-equilibrium
C             (TH /= TE /= TV(1) /= TV(2) /= ... )
C
C     -Obtain RHO, RHOI, Y, RHOE, RHOEE, RHOEV from CFD code
C     -Compute TE, TR, TV(I)
C     -Use temperatue and specie mass fractions from previous
C      iteration as initial guess
C-----------------------------------------------------------------------
      CALL TCNEQNTINEWTONF(RHOE, RHOEE, RHOEV, RHOI, TINI, TH, TE, TV)
      CALL MASS2PRESSUREF(RHO, TH, TE, Y, P)
      CALL MASS2MOLEF(Y, X)
      CALL NUMBERDF(P, TH, TE, X, ND)

      TR = TH
      TVEC(1) = TH
      DO IVIB = 1, NVIB
        TVEC(1+IVIB) = TV(1)
      ENDDO
      TVEC(NVIB+2) = TE

      CALL ARRHENIUSF(Y, TOLY, P, TVEC, RHO, OMEGA)


C-----------------------------------------------------------------------
C     Caloric Equation of State (EOS)
C-----------------------------------------------------------------------
      THP = TH*EPSP1
      TEP = TE*EPSP1
      TRP = TR*EPSP1
      IF( NV == 0 ) THEN
        TVP(1) = TV(1)*EPSP1
      ELSE
        DO IV = 1, NV
          TVP(IV) = TV(IV)*EPSP1
        ENDDO
      ENDIF
```

```
      CALL ENERGYMASSF(TH, TE, TR, TV, P,
     &                 EM1, EM2, EM3, EM4, EM5, EM6)
      CALL ENERGYMASSF(THP, TEP, TRP, TVP, P,
     &                 EM1P, EM2P, EM3P, EM4P, EM5P, EM6P)

      DO IS = 1, NS
        CPE       = (EM3P(IS) - EM3(IS))/(EPS*TH)
        CPR       = (EM4P(IS) - EM4(IS))/(EPS*TH)
        CPV       = (EM5P(IS) - EM5(IS))/(EPS*TH)
        CPINT(IS) = MMI(IS)*(CPE + CPR + CPV)
      ENDDO


C-----------------------------------------------------------------------
C       TRANSPORT equations
C-----------------------------------------------------------------------
      CALL COMPOTOLF(X, TOLX, XTOL)
      CALL COLLISIONF(TH, TE, ND, X)
      CALL ETACGF(XTOL, ETA)
      CALL LAMBDAINTF(CPINT, XTOL, LAMBDAINTH)
      CALL LAMBDACHICGF(XTOL, LAMBDATH, CHIH)

      IF (NE /= 0) THEN
        CALL TDIFEF(XTOL, TE, CHIE, 3)
        CALL LAMBDAEF(XTOL, TE, LAMBDATE, 3)
      ELSE
        LAMBDATE = 0.D0
        DO IS = 1, NS
          CHIE(IS) = 0.D0
        ENDDO
      ENDIF

      DO IS = 1, NS
      DF(IS) = (XP(IS) - X(IS))/(TH*EPS) + (CHIE(IS) + CHIH(IS))/TH
      ENDDO

C-----------------------------------------------------------------------
C       Diffusion fluxes
C-----------------------------------------------------------------------
      CALL CORRECTION (XTOL, 1, FIJ)
      IF (NE /= 0) THEN
        CALL SMRAMCG (XTOL, TH, TE, ND, DF, FIJ, JDIF, EAMB)
      ELSE
        CALL SMNEUTCG (XTOL, ND, DF, FIJ, JDIF)
        EAMB = 0.D0
      ENDIF
      CALL DIJFICK (XTOL, ND, DIJ)
      DO IS = 1, NS
        JDIF2(IS) = 0.D0
```

```
            DO JS = 1, NS
              JDIF2(IS) = JDIF2(IS) -RHOI(IS) *DIJ(IS,JS) *DF(JS)
            ENDDO
          ENDDO

C-----------------------------------------------------------------------
C         Total thermal conductivity
C-----------------------------------------------------------------------
          LAMBDAR = 0.D0; LAMBDAR2 = 0.D0
          DO IS = 1, NS
            LAMBDAR  = LAMBDAR  -(EM1(IS) +2.D0/3.D0 *EM2(IS))* JDIF(IS)
            LAMBDAR2 = LAMBDAR2 -(EM1(IS) +2.D0/3.D0 *EM2(IS))* JDIF2(IS)
          ENDDO

          CALL KCONDUCTIVITY (P, TH, X, XTOL, XN, EPS, LAMBDATOT)

          LAMBDATOT2 = LAMBDAINTH + LAMBDATH + LAMBDATE + LAMBDAR

          CALL KCONDUCTIVITYNEQ (P, TH, X, XTOL, XN, EPS, LAMBDATOT3)


C-----------------------------------------------------------------------
C         Deallocate memory
C-----------------------------------------------------------------------
          CALL FINALIZEF()

C-----------------------------------------------------------------------
          STOP

C-----------------------------------------------------------------------
          END
C-----------------------------------------------------------------------
```

A.2 C implementation

```
#include <stdio.h>
#include <stdlib.h>


char path[] = "../mutation/resources";
char mixture[] = "air5";
char reaction[] = "air5";
char transfer[] = "empty";


int IMOD;
int IS,JS,IC,IV,IE,IVIB,IELEQ;
int NS,NC,NV,NE,NVIB,NELEQ;

double EPS, EPSP1;
double TOLX, TOLY;
double *XTOL;
double RHO, RHOE, RHOEE, RHOER, *RHOEV;
double MM, *MMI;
double *X, *XP, *XN;
double *Y, *Y3, *Y4, *Y5;
double *XINI, *YINI;
double *RHOI;
double P, ND;
double T, T2, T3, TINI;
double TH , TE , TR , *TV ;
double THP, TEP, TRP, *TVP;
double *TVEC;
double EM, HM;
double *EM1 , *EM2 , *EM3 , *EM4 , *EM5 , *EM6 ;
double *EM1P, *EM2P, *EM3P, *EM4P, *EM5P, *EM6P;
double *HM1 , *HM2 , *HM3 , *HM4 , *HM5 , *HM6 ;
double *OMEGA;
double ETA;
double CPE, CPR, CPV;
double *CPINT;
double LAMBDAINTH, LAMBDATH, LAMBDATE;
double *CHIE, *CHIH;
double *FIJ;
double *JDIF;
double *DF;


int main(int argc, char** argv)
{
    IMOD = 0;
```

```
    EPS    = 1.e-2;
    EPSP1 = 1.e0 + EPS;

    TOLX = 1.e-16;
    TOLY = 1.e-20;

//-----------------------------------------------------------------------
    length(path, mixture, reaction, transfer);

    NS     = getns_();
    NC     = getnc_();
    NE     = getne_();
    NV     = getnv_();
    NVIB   = getnvib_();
    NELEQ = getneleq_();

//-----------------------------------------------------------------------
    X      = (double *)calloc(NS, sizeof(double));
    Y      = (double *)calloc(NS, sizeof(double));
    XP     = (double *)calloc(NS, sizeof(double));
    Y3     = (double *)calloc(NS, sizeof(double));
    Y4     = (double *)calloc(NS, sizeof(double));
    Y5     = (double *)calloc(NS, sizeof(double));
    XN     = (double *)calloc(NC, sizeof(double));
    MMI    = (double *)calloc(NS, sizeof(double));
    XINI   = (double *)calloc(NS, sizeof(double));
    YINI   = (double *)calloc(NS, sizeof(double));
    XTOL   = (double *)calloc(NS, sizeof(double));
    RHOI   = (double *)calloc(NS, sizeof(double));
    CPINT = (double *)calloc(NS, sizeof(double));
    TV     = (double *)calloc(NV, sizeof(double));
    TVP    = (double *)calloc(NV, sizeof(double));
    TVEC   = (double *)calloc(NVIB+2, sizeof(double));

    EM1   = (double *)calloc(NS, sizeof(double));
    EM2   = (double *)calloc(NS, sizeof(double));
    EM3   = (double *)calloc(NS, sizeof(double));
    EM4   = (double *)calloc(NS, sizeof(double));
    EM5   = (double *)calloc(NS, sizeof(double));
    EM6   = (double *)calloc(NS, sizeof(double));
    HM1   = (double *)calloc(NS, sizeof(double));
    HM2   = (double *)calloc(NS, sizeof(double));
    HM3   = (double *)calloc(NS, sizeof(double));
    HM4   = (double *)calloc(NS, sizeof(double));
    HM5   = (double *)calloc(NS, sizeof(double));
    HM6   = (double *)calloc(NS, sizeof(double));
    EM1P = (double *)calloc(NS, sizeof(double));
    EM2P = (double *)calloc(NS, sizeof(double));
    EM3P = (double *)calloc(NS, sizeof(double));
```

```
    EM4P = (double *)calloc(NS, sizeof(double));
    EM5P = (double *)calloc(NS, sizeof(double));
    EM6P = (double *)calloc(NS, sizeof(double));

    OMEGA = (double *)calloc(NS, sizeof(double));
    CHIH  = (double *)calloc(NS, sizeof(double));
    CHIE  = (double *)calloc(NS, sizeof(double));
    DF    = (double *)calloc(NS, sizeof(double));
    JDIF  = (double *)calloc(NS, sizeof(double));
    CHIH  = (double *)calloc(NS, sizeof(double));
    CHIE  = (double *)calloc(NS, sizeof(double));
    FIJ   = (double *)calloc(NSMAX*(NSMAX-1)/2), sizeof(double));

//-----------------------------------------------------------------------
    TE = TH;
    TR = TH;
    if( NV == 0 )
    {   TV[0] = 1.;
    }
    else
    {   for(IV=0;IV<NV;IV++)
        {   TV[IV] = TH;
        }
    }

    TVEC[0] = TH;
    for(IVIB=0;IVIB<NVIB;IVIB++)
    {   TVEC[IVIB+1] = TV[0];
    }
    TVEC[NVIB+1] = TE;

//-----------------------------------------------------------------------
    THP = TH*EPSP1;
    TEP = TE*EPSP1;
    TRP = TR*EPSP1;
    if( NV == 0 )
    {   TVP[0] = TV[0]*EPSP1;
    }
    else
    {   for(IV=0;IV<NV;IV++)
        {   TVP[IV] = TV[IV]*EPSP1;
        }
    }


//-----------------------------------------------------------------------
//       Subroutines to be called only once
//-----------------------------------------------------------------------
    initialize(IMOD);
```

```
    setmass(MMI);
    nuclear(XN);


//-------------------------------------------------------------------------
//      CASE 1: Local thermodynamic equilibrium (LTE)
//              with constant elemental fractions
//
//      -Obtain RHO, RHOE from CFD code
//      -Use temperatue and specie mass fractions from previous
//       iteration as initial guess
//-------------------------------------------------------------------------
    tceqnewton(&RHOE, &RHO, XN, YINI, &TINI, &T, Y);
    mass2pressure(&RHO, &T, &T, Y, &P);
    mass2mole(Y, X);
    numberd(&P, &T, &T, X, &ND);

    TH = T; TE = TH; TR = TH;
    TVEC[0] = TH;
    TVEC[NVIB+2-1] = TE;


//-------------------------------------------------------------------------
//      CASE 2: Chemical non-equilibrium and
//              local thermal equilibrium (LTE)
//
//      -Obtain RHO, RHOI, Y, RHOE from CFD code
//      -Use temperatue and specie mass fractions from previous
//       iteration as initial guess
//-------------------------------------------------------------------------
    tcneqnewton(&RHOE, RHOI, &TINI, &T);
    mass2pressure(&RHO, &T, &T, Y, &P);
    mass2mole(Y, X);
    numberd(&P, &T, &T, X, &ND);

    TH = T; TE = TH; TR = TH;
    TVEC[0] = TH;
    TVEC[NVIB+2-1] = TE;

    arrhenius(Y, &TOLY, &P, TVEC, &RHO, OMEGA);


//-------------------------------------------------------------------------
//      CASE 3: Chemical and thermal non-equilibrium
//              (TH != TE != TV(1) != TV(2) != ... )
//
//      -Obtain RHO, RHOI, Y, RHOE, RHOEE, RHOEV from CFD code
//      -Compute TE, TR, TV(I)
//      -Use temperatue and specie mass fractions from previous
```

```
//        iteration as initial guess
//-------------------------------------------------------------------------
    tcneqnntewton(&RHOE, &RHOEE, RHOEV, &RHOI, &TINI, &TH, &TE, TV);
    mass2pressure(&RHO, &TH, &TE, Y, &P);
    mass2mole(Y, X);
    numberd(&P, &TH, &TE, X, &ND);

    TR = TH;
    TVEC[0] = TH;
    for(IVIB=0;IVIB<NVIB;IVIB++)
    {   TVEC[IVIB+2-1] = TV[IVIB];
    }
    TVEC[NVIB+1] = TE;

    arrhenius(Y, &TOLY, &P, TVEC, &RHO, OMEGA);


//-------------------------------------------------------------------------
//        Caloric Equation of State (EOS)
//-------------------------------------------------------------------------
    THP = TH*EPSP1;
    TEP = TE*EPSP1;
    TRP = TR*EPSP1;
    if( NV == 0 )
    {   TVP[0] = TV[0]*EPSP1;
    }
    else
    {   for(IV=0;IV<NV;IV++)
        {   TVP[IV] = TV[IV]*EPSP1;
        }
    }

    energymass(&TH , &TE , &TR , TV , &P, EM1 , EM2 , EM3 , EM4 , EM5 , EM6 );
    energymass(&THP, &TEP, &TRP, TVP, &P, EM1P, EM2P, EM3P, EM4P, EM5P, EM6P);

    for(IS=0;IS<NS;IS++)
    {   CPE       = (EM3P[IS] - EM3[IS])/(EPS*TH);
        CPR       = (EM4P[IS] - EM4[IS])/(EPS*TH);
        CPV       = (EM5P[IS] - EM5[IS])/(EPS*TH);
        CPINT[IS] = MMI[IS]*(CPE + CPR + CPV);
    }


//-------------------------------------------------------------------------
//        TRANSPORT equations
//-------------------------------------------------------------------------
    compotol(X, &TOLX, XTOL);
    collision(&TH, &TE, &ND, X);
    etacg(XTOL, &ETA);
```

```
    lambdaint(CPINT, XTOL, &LAMBDAINTH);
    lambdachicg(XTOL, &LAMBDATH, CHIH);

    if( NE != 0 )
    {   tdife(XTOL, &TE, CHIE, 3);
        lambdae(XTOL, &TE, &LAMBDATE, 3);
    }
    else
    {   LAMBDATE = 0.;
        for(IS=0;IS<NS;IS++)
        {   CHIE[IS] = 0.;
        }
    }

    for(IS=0;IS<NS;IS++)
    {   DF[IS] = (XP[IS] - X[IS])/(TH*EPS) + (CHIE[IS] + CHIH[IS])/TH;
    }

//-----------------------------------------------------------------------
//      Diffusion fluxes
//-----------------------------------------------------------------------
    correction(XTOL, 1, FIJ);
    if( NE != 0 )
    {   smramcg(XTOL, &TH, &TE, &ND, DF, FIJ, JDIF, &EAMB);
    }
    else
    {   smneutcg(XTOL, &ND, DF, FIJ, JDIF);
        EAMB = 0.;
    }
    dijfick(XTOL, &ND, DIJ);
    for(IS=0;IS<NS;IS++)
    {   JDIF(IS) = 0.;
        for(JS=0;JS<NS;JS++)
        {   JDIF[IS] = JDIF[IS] - RHOI[IS]*DIJ[IS][JS]*DF[JS];
        }
    }

//-----------------------------------------------------------------------
//      Total thermal conductivity
//-----------------------------------------------------------------------
    LAMBDAR = 0.;
    for(IS=0;IS<NS;IS++)
    {   LAMBDAR = LAMBDAR - (EM1[IS] + 2./3.*EM2[IS])*JDIF[IS];
    }

    kconductivity(&P, &TH, X, XTOL, XN, &EPS, &LAMBDATOT);

    LAMBDATOT = LAMBDAINTH + LAMBDATH + LAMBDATE + LAMBDAR;
```

```
    kconductivityneq(&P, &TH, X, XTOL, XN, &EPS, &LAMBDATOT);

//-----------------------------------------------------------------------
//        Deallocate memory
//-----------------------------------------------------------------------
    finalize();

//-----------------------------------------------------------------------

}
```

A.3 Java implementation

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */


import mutation.Common;
import static mutation.Interface.*;

/**
 *
 */
public class UseMutation
{

    String PATH = "../mutation/resources";
    String MIXTURE = "air5";
    String REACTION = "air5";
    String TRANSFER = "empty";


    int IMOD;
    int IS,JS,IC,IV,IE,IVIB,IELEQ;
    int NS,NC,NV,NE,NVIB,NELEQ;

    double EPS, EPSP1;
    double TOLX, TOLY;
    double XTOL[];
    double RHO, RHOE, RHOEE, RHOER, RHOEV[];
    double MM, MMI[];
    double XN[];
    double XP[];
    double X[], Y[], Y3[], Y4[], Y5[];
    double XINI[], YINI[];
    double RHOI[];
    double P, ND;
    double T, TINI, T2, T3;
    double TH , TE , TR , TV [];
    double THP, TEP, TRP, TVP[];
    double TVEC[];
    double EM, HM;
    double EM1 [], EM2 [], EM3 [], EM4 [], EM5 [], EM6 [];
    double EM1P[], EM2P[], EM3P[], EM4P[], EM5P[], EM6P[];
    double HM1 [], HM2 [], HM3 [], HM4 [], HM5 [], HM6 [];
    double OMEGA[];
    double ETA;
    double CPE, CPR, CPV;
```

```
double CPINT[];
double LAMBDAINTH, LAMBDATH, LAMBDATE;
double CHIE[], CHIH[];
double JDIF[], JDIF2[], DF[], FIJ[];
double EAMB;
double DIJ[][];
double LAMBDAR, LAMBDAR2, LAMBDATOT, LAMBDATOT2, LAMBDATOT3;


Common common;
Common.Memory1 memory1;
Common.Memory2 memory2;
Common.Memory3 memory3;
Common.Memory4 memory4;
Common.Memory5 memory5;
Common.Memory6 memory6;



public UseMutation(double P, double T)
{
    common = new Common();
    memory1 = common.new Memory1();
    memory2 = common.new Memory2();
    memory3 = common.new Memory3();
    memory4 = common.new Memory4();
    memory5 = common.new Memory5();
    memory6 = common.new Memory6();

    IMOD = 0;

    EPS   = 1.e-02;
    EPSP1 = 1.e+00 + EPS;

    TOLX = 1.e-16;
    TOLY = 1.e-20;

    this.P  = P;
    this.TH = T;


    length(PATH, MIXTURE, REACTION, TRANSFER);
    initialize(IMOD);

    getMemory1(memory1);
    NS = memory1.NS;
    NC = memory1.NC;
    NE = memory1.NE;
    NV = memory1.NV;
```

```
NVIB = memory1.NVIB;

X = new double[NS];
Y = new double[NS];
XP = new double[NS];
Y3 = new double[NS];
Y4 = new double[NS];
Y5 = new double[NS];
XN = new double[NC];
MMI = new double[NS];
XINI = new double[NS];
YINI = new double[NS];
XTOL = new double[NS];
RHOI = new double[NS];
CPINT = new double[NS];
RHOEV = new double[NV];
TV = new double[NV];
TVP = new double[NV];
TVEC = new double[NVIB+2];

EM1 = new double[NS];
EM2 = new double[NS];
EM3 = new double[NS];
EM4 = new double[NS];
EM5 = new double[NS];
EM6 = new double[NS];
HM1 = new double[NS];
HM2 = new double[NS];
HM3 = new double[NS];
HM4 = new double[NS];
HM5 = new double[NS];
HM6 = new double[NS];
EM1P = new double[NS];
EM2P = new double[NS];
EM3P = new double[NS];
EM4P = new double[NS];
EM5P = new double[NS];
EM6P = new double[NS];

OMEGA = new double[NS];
CHIH = new double[NS];
DF = new double[NS];
FIJ = new double[NS*(NS-1)/2];
DIJ = new double[NS][NS];

//-------------------------------------------------------------------------

TE = TH;
TR = TH;
```

```
        if( NV == 0 )
        {   TV[0] = 1.;
        }
        else
        {   for(IV=0;IV<NV;IV++)
            {   TV[IV] = TH;
            }
        }

        TVEC[0] = TH;
        for(IVIB=0;IVIB<NVIB;IVIB++)
        {   TVEC[IVIB+1] = TV[0];
        }
        TVEC[NVIB+1] = TE;

//-------------------------------------------------------------------------
        THP = TH*EPSP1;
        TEP = TE*EPSP1;
        TRP = TR*EPSP1;
        if( NV == 0 )
        {   TVP[0] = TV[0]*EPSP1;
        }
        else
        {   for(IV=0;IV<NV;IV++)
            {   TVP[IV] = TV[IV]*EPSP1;
            }
        }


//-------------------------------------------------------------------------
//      Subroutines to be called only once
//-------------------------------------------------------------------------
        initialize(IMOD);
        setmass(MMI);
        nuclear(XN);


//-------------------------------------------------------------------------
//      CASE 1: Local thermodynamic equilibrium (LTE)
//              with constant elemental fractions
//
//      -Obtain RHO, RHOE from CFD code
//      -Use temperatue and specie mass fractions from previous
//       iteration as initial guess
//-------------------------------------------------------------------------
        T = tceqnewton(RHOE, RHO, XN, YINI, TINI, T, Y);
        P = mass2pressure(RHO, T, T, Y, P);
        mass2mole(Y, X);
        ND = numberd(P, T, T, X, ND);
```

```
        TH = T; TE = TH; TR = TH;
        TVEC[0] = TH;
        TVEC[NVIB+2-1] = TE;


//-------------------------------------------------------------------------
//      CASE 2: Chemical non-equilibrium and
//              local thermal equilibrium (LTE)
//
//      -Obtain RHO, RHOI, Y, RHOE from CFD code
//      -Use temperatue and specie mass fractions from previous
//       iteration as initial guess
//-------------------------------------------------------------------------
        T = tcneqnewton(RHOE, RHOI, TINI, T);
        P = mass2pressure(RHO, T, T, Y, P);
        mass2mole(Y, X);
        ND = numberd(P, T, T, X, ND);

        TH = T; TE = TH; TR = TH;
        TVEC[0] = TH;
        TVEC[NVIB+2-1] = TE;

        arrhenius(Y, TOLY, P, TVEC, RHO, OMEGA);


//-------------------------------------------------------------------------
//      CASE 3: Chemical and thermal non-equilibrium
//              (TH != TE != TV(1) != TV(2) != ... )
//
//      -Obtain RHO, RHOI, Y, RHOE, RHOEE, RHOEV from CFD code
//      -Compute TE, TR, TV(I)
//      -Use temperatue and specie mass fractions from previous
//       iteration as initial guess
//-------------------------------------------------------------------------
        tcneqntnewton(RHOE, RHOEE, RHOEV, RHOI, TINI, TH, TE, TV, TVEC);

        TH = TVEC[0];   TR = TH;
        for(IVIB=0;IVIB<NVIB;IVIB++)
        {    TV[IVIB] = TVEC[IVIB+2-1];
        }
        TE = TVEC[NVIB+1];

        P = mass2pressure(RHO, TH, TE, Y, P);
        mass2mole(Y, X);
        ND = numberd(P, TH, TE, X, ND);
        arrhenius(Y, TOLY, P, TVEC, RHO, OMEGA);
```

```
//-------------------------------------------------------------------------
//        Caloric Equation of State (EOS)
//-------------------------------------------------------------------------
        THP = TH*EPSP1;
        TEP = TE*EPSP1;
        TRP = TR*EPSP1;
        if( NV == 0 )
        {   TVP[0] = TV[0]*EPSP1;
        }
        else
        {   for(IV=0;IV<NV;IV++)
            {   TVP[IV] = TV[IV]*EPSP1;
            }
        }
        energymass(TH , TE , TR , TV , P, EM1 , EM2 , EM3 , EM4 , EM5 , EM6 );
        energymass(THP, TEP, TRP, TVP, P, EM1P, EM2P, EM3P, EM4P, EM5P, EM6P);

        for(IS=0;IS<NS;IS++)
        {   CPE         = (EM3P[IS] - EM3[IS])/(EPS*TH);
            CPR         = (EM4P[IS] - EM4[IS])/(EPS*TH);
            CPV         = (EM5P[IS] - EM5[IS])/(EPS*TH);
            CPINT[IS] = MMI[IS]*(CPE + CPR + CPV);
        }


//-------------------------------------------------------------------------
//        TRANSPORT equations
//-------------------------------------------------------------------------
        compotol(X, TOLX, XTOL);
        collision(TH, TE, ND, X);
        ETA = etacg(XTOL, ETA);
        LAMBDAINTH = lambdaint(CPINT, XTOL, LAMBDAINTH);
        lambdachicg(XTOL, LAMBDATH, CHIH);

        if( NE != 0 )
        {   tdife(XTOL, TE, CHIE, 3);
            LAMBDATE = lambdae(XTOL, TE, LAMBDATE, 3);
        }
        else
        {   LAMBDATE = 0.;
            for(IS=0;IS<NS;IS++)
            {   CHIE[IS] = 0.;
            }
        }

        for(IS=0;IS<NS;IS++)
        {   DF[IS] = (XP[IS] - X[IS])/(TH*EPS) + (CHIE[IS] + CHIH[IS])/TH;
        }
```

```
//-------------------------------------------------------------------------
//        Diffusion fluxes
//-------------------------------------------------------------------------
    correction(XTOL, 1, FIJ);
    if( NE != 0 )
    {   EAMB = smramcg(XTOL, TH, TE, ND, DF, FIJ, JDIF, EAMB);
    }
    else
    {   smneutcg(XTOL, ND, DF, FIJ, JDIF);
        EAMB = 0.;
    }
    dijfick(XTOL, ND, DIJ);
    for(IS=0;IS<NS;IS++)
    {   JDIF[IS] = 0.;
        for(JS=0;JS<NS;JS++)
        {   JDIF[IS] = JDIF[IS] - RHOI[IS]*DIJ[IS][JS]*DF[JS];
        }
    }

//-------------------------------------------------------------------------
//        Total thermal conductivity
//-------------------------------------------------------------------------
        LAMBDAR = 0.;
        for(IS=0;IS<NS;IS++)
        {   LAMBDAR = LAMBDAR - (EM1[IS] + 2./3.*EM2[IS])*JDIF[IS];
        }

        LAMBDATOT = kconductivity(P, TH, X, XTOL, XN, EPS, LAMBDATOT);

        LAMBDATOT = LAMBDAINTH + LAMBDATH + LAMBDATE + LAMBDAR;

        LAMBDATOT = kconductivityneq(P, TH, X, XTOL, XN, EPS, LAMBDATOT);


//-------------------------------------------------------------------------
//        Deallocate memory
//-------------------------------------------------------------------------
        finalizej();

    }
}
```